Student Name: **Student id:** **Sect: #:**

QUESTION #	1	2	3	4	5	6	TOTAL
MAX POINTS	13	13	13	12	12	12	75
POINTS EARNED							

University of Bahrain College of Information Technology

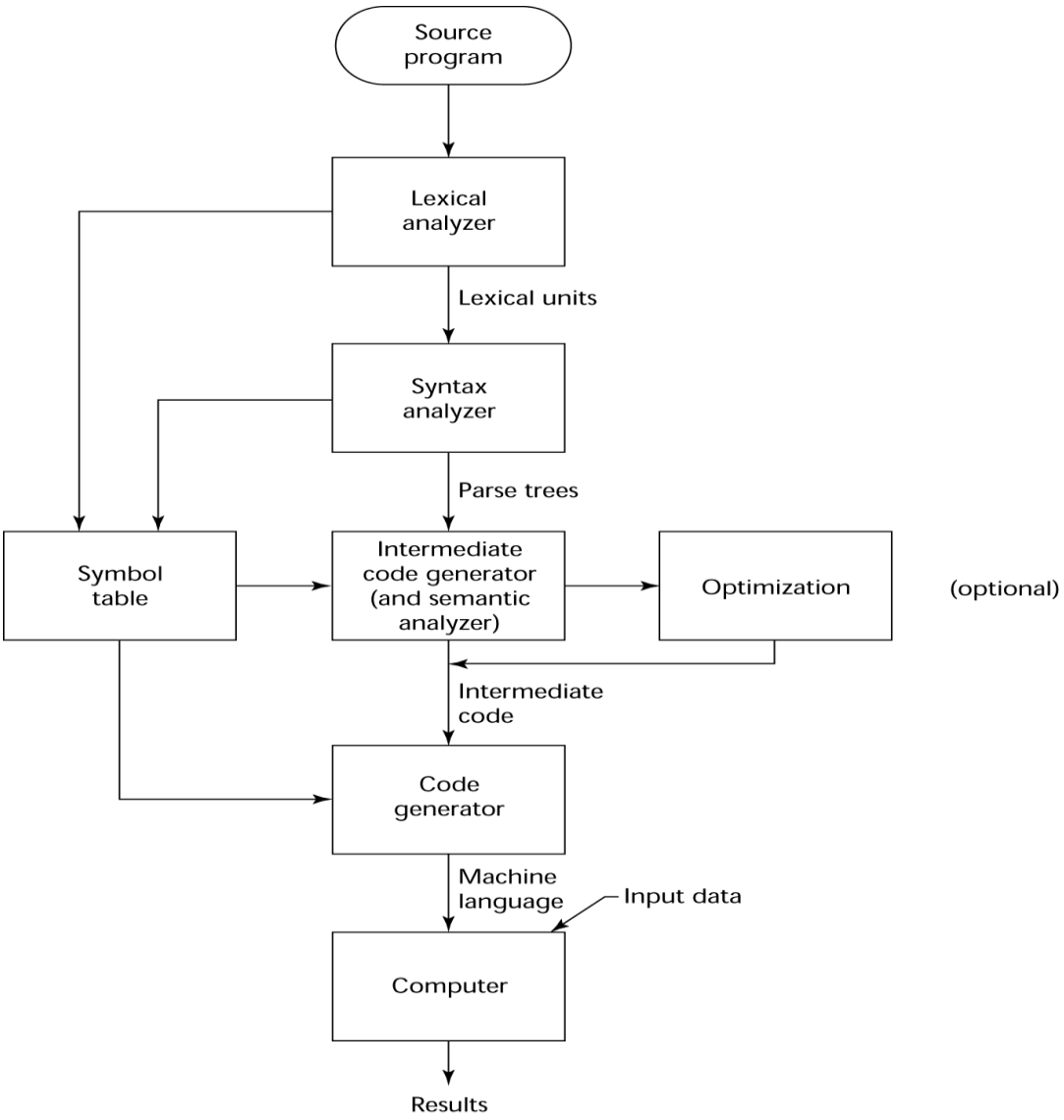
Department of Computer Science

ITCS 332: Organization of Programming Languages FIRST TEST Date: OCT 29, 2015

****

QUESTION *ONE*: [13 pts]

Draw the flowchart that illustrates the steps of a compilation process of any HLL programming language .



QUESTION TWO: **[13 pts]**

- Convert each of the following BNF rules into ONE equivalent EBNF rule

1) <int> → <digits>| <sign> <digits>
<digits> → <digit>| <digits> <digit>
<sign> → +| -

<int> → [(+|-)] <digit> { <digit> }

2) <cat> → <cat> *"*"* <dog> | <cat> *"/"* <dog>
| <cat> *"%"* <dog> | <dog>

<cat> → <dog> { (* | / | %) <dog> }

3) *<real>* → *<sign> <int> "." <int> "E" <int>* | *<sign> <int> "." <int>*
| *<sign> <int> "." <int> "E" <sign> <int>*
<int> → *<dig>* | *<int> <dig>*
<sign> → *"+"* | *"-"*

<real> → (+|-) <dig>{<dig>} "." <dig>{<dig>} ["E" [(+|-)] <dig>{<dig>}]

- **Fill in blanks**

1) Rewrite rule <cat> → <cat> … given above, so that operators * / % associate right to left.

<cat> → <dog> "*" <cat> | <dog> "/" <cat>
| <dog> "%" <cat> | <dog

2) In axiomatic semantics, each statement is preceded by a **precondition** and followed by a **postcondition**.

3) An attribute grammar is a context-free grammar (BNF) plus 3 additions, name any two of them:
SEMANTIC FUNCTIONS, PREDICATE FUNCTIONS, ATTRIBUTE VALUES.

4) Name two issues (examples) in programming languages that cannot be described by BNF rules:

- **Type compatibility rules.**
- **Identifiers must be declared before referenced.**

QUESTION THREE: **[13 pts]**

- **Fill in blanks**

1) The number of tokens in a C++ statement: **`while(!((dt-23*ct/17)<76)) dt=5;`** is **12** and the number lexemes of is **21**.
2) The weakest precondition for each kind of statements can be computed by **an axiom** or by **an inference rule**.
3) Ambiguities in BNF grammars of arithmetic expression can be solved by introducing **precedence rules** and **assosiativity rule** into its grammar.
4) In denotational semantics, the state changes are defined by **MATHEMATICAL FUNCTIONS**.
   In operational semantics, the state changes are defined by **CODED ALGORITHMS**.

- Convert each of the following EBNF rule into equivalent BNF rule(s).

1) `<musa>` → `<var> { ( $$ | ## ) <var> }`

```
<musa>   → <var>
         | <var> $$ <musa>
         | <var> ## <musa>
```

2) `<SEQSTR>` → `<STRING> # { <STRING> # }`

```
<SEQSTR>  → <STRING> #
          | <STRING> # < SEQSTR >
```

3) `<comas>` → `<var> (+=|-=|*=|/=|%=) [(+|-)] <int>`

```
<comas>   → <var> <op> <int>
          | <var> <op> <sign> <int>

<op>      → += | -= | *= | /= | %=
<sign>    → + | -
```

QUESTION FOUR: **[12 pts]**

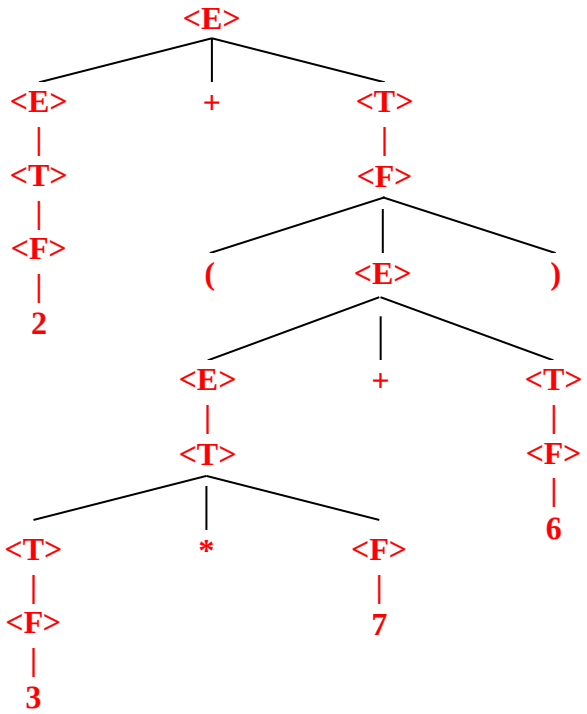
Carefully study the following grammar and answer the next two questions.

<E> → <E> + < T> | <E > – < T> |<T>
<T> → <T> * <F> | <T> / <F> | <F>
<F> → (<E>) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Construct the rightmost derivation of the sentence : 2 + (3 * 7 + 6)

```
<E>  =>  <E> + <T>
     =>  <E> + <F>
     =>  <E> + ( <E> )
     =>  <E> + ( <E>          + <T> )
     =>  <E> + ( <E>          + <F> )
     =>  <E> + ( <E>          +  6  )
     =>  <E> + ( <T>          +  6  )
     =>  <E> + ( <T> * <F>    +  6  )
     =>  <E> + ( <T> *  7     +  6  )
     =>  <E> + ( <F> *  7     +  6  )
     =>  <E> + (  3  *  7     +  6  )
     =>  <T> + (  3  *  7     +  6  )
     =>  <F> + (  3  *  7     +  6  )
     =>   2  + (  3  *  7     +  6  )
```

- Construct the parse tree of the sentence : 2 + (3 * 7 + 6)



QUESTION FIVE: **[12 pts]**

1) A floating-point value <float> is a sequence of a sign (+ or −) followed by a sequence of digits followed by **dot** followed by a sequence of digits. Plus sign is optional. Examples of accepted values: +20.34567,-7.6, 307.80,-0.2, …. Write ONE EBNF rule to describe the <float> values.

<float> → [(+|-)] <digit> {<digit} . <digit> {<digit>}

- The increment/decrement operator in C++ <inc_dec> is a standalone statement consisting of a variable name followed or preceded by ++ or -- such as: a++, ++a, a--, --a. The variable name <id> is a lower-case letter <Let> followed by zero or more letters or digits <dig>. Construct the EBNF rules for the increment/decrement operator in C++ as described above.

<inc_dec> → <id>(++|--) | (--|++)<id>

<id> → <Let> {(<Let> | <dig>)}

- Construct BNF rules that describe the logical expression <LE>. A logical expression consists of one or more boolean terms <b_term> separated by a logical operator ||. A boolean term consists of one or more boolean factors <b_factor> separated by a logical operator && which has higher precedence than ||. A boolean factor may be a variable <var>, relational expression <rel_exp>, or a logical expression enclosed between parentheses. A boolean factor may be preceded by not operator !. A relational expression consists of two arithmetic expressions <AE> separated by a relational operator <RO>: **==, *!*=.** All relational operators have equal precedence. Logical operators have lower precedence than relational operators. Assume arithmetic expression is already defined.

<LE> → <b_term> | <LE> || <b_term>

<b_term> → <b_factor> | <b_term> && <b_factor>

<b_factor> → <var> | <rel_exp> | (<LE>) | ! <LE>

<rel_exp> → <AE> <RO> <AE>

<RO> → == | !=

QUESTION *SIX*: Fill in blanks [12 pts]

1) Give two suggestions to improve the reliability of C++:

- **Perform index range checking.**
- **Remove pointers.**

2) The two main features of imperative languages are:

- **Algorithms of imperative languages consist of detailed ordered steps.**
- **Imperative languages are based on variables, assignment and iteration**.

3) Give two suggestions to reduce the execution times of programs in any language without affecting its reliability:

- **Reduce degree of optimization.**
- **Use macros instead of subprograms.**

4) Give two disadvantages of pure interpretation systems:

- **Pure interpretation systems are slow in execution.**
- **Require large memory space to store source program and symbol table.**

5) Give two reasons why computers do not use HLL as a machine language:

- **Designing computers that use HLL as a machine language makes computers complex and expensive.**
- **Not flexible (Language-dependent).**

6) Give two advantages of using preprocessor macros:

- **Programs with macros are generic (typeless operands).**
- **Programs with macros execute faster than those with subprograms.**